Modelowanie procedur przetwarzania rozproszonego w sieciach informatycznych laboratoriów dydaktycznych.

Jarosław Szymańda, Maciej Noszczyński
Politechnika Wrocławska, Instytut Podstaw Elektrotechniki i Elektrotechnologii
50 - 370 Wrocław, ul. Wybrzeże Wyspiańskiego 27, E-mail: jaroslaw.szymanda@pwr.wroc.pl

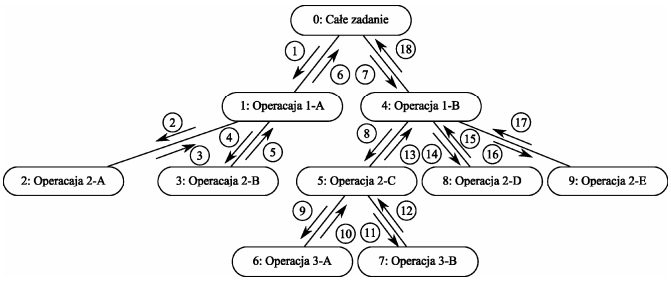
Streszczenie - W referacie przedstawiono adaptację dedykowanego systemu przetwarzania rozproszonego, przeznaczonego do wdrożenia w studenckich laboratoriach-sieciowo komputerowych wydziałów elektrycznych ze szczególnym uwzględnieniem zagadnień z zakresu kursów z cyfrowego przetwarzania sygnałów oraz metod numerycznych w technice. Podstawowym założeniem projektu było przyjęcie formuły maksymalnej zwięzłości semantyki logicznej oraz przenośności oprogramowania pomiędzy różnymi systemami operacyjnymi, uniezależniając tym samym meritum zagadnienia dydaktycznego od platformy systemowo-sprzętowej.

I. Wstęp

Prezentowany pakiet składa się z zestawu programów sieciowych, działających na zasadzie peer-to-peer (równy-dorównego) jako „goście" w przestrzeniach systemów operacyjnych użytkownika. Możliwość uruchomienia odpowiednich komponentów systemu na dowolnej liczbie komputerów jest podstawą przyjętego modelu przetwarzania rozproszonego. Przetwarzanie rozproszone w tym przypadku polega na udostępnianiu dowolnych zasobów infrastruktury (moc obliczeniowa, dostęp do urządzeń wejścia/wyjścia, pamięci masowej, itp.) poprzez komunikację w lokalnych (LAN) oraz rozległych (WAN) sieciach komputerowych. Przedstawiona propozycja umożliwia zestawienie np. bardzo wydajnego wirtualnego komputera z wielu mniej wydajnych jednostek (idea tzw. klastrów obliczeniowych), lub zrealizowanie zadania z zakresu automatyki kontrolno-pomiarowej i sterowania - kiedy program uruchomiony na jednym z komputerów reaguje na sygnały urządzeń wejścia/wyjścia na innym. W skład systemu wchodzą również: sieciowy magazyn danych oraz serwer nadzorujący wykonywanie procedur. Magazyn udostępnia swoim klientom polecenia składowania wyników przetwarzania w ramach „swoich" zasobów pamięci operacyjnej, identyfikując je poprzez ścieżki dostępu składające się z hierarchicznie zależnych nazw alokowanych w nim w czasie rzeczywistym. Serwer nadzorujący natomiast odpowiada za udostępnianie odpowiednich procedur pomocniczych, informujących każdego „klienta" o ilości, lokalizacji i statusie serwerów końcowych[1] realizujących powierzone im procedury wykonawcze. Do opisu schematu działania programu rozwiązującego dany problem, wykorzystano język symboliczny oparty o język programowania Python, opracowany przez Guido van Rossuma w Stichting Matematisch Centrum na początku lat 90-tych [1]. Założona realizacja algorytmów w wariantach o jak największej niezależności systemowo-sprzętowej umożliwia równoległe zaprogramowanie eksperymentu w środowiskach obejmujących systemy operacyjne oraz architekturę procesorów takich jak: Windows (na architekturach IA-32, IA-64, Alpha), Windows CE, DOS, Macintosh OS X, Linux (na bardzo wielu architekturach np: Sun SPARC, Alpha, Motorola/IBM PowerPC, ARM, MIPS, HP PA-RISC, IA-64, S/390), OS/2, Playstation, QNX, Risc OS, Sparc Solaris, VMS) oraz potencjalnie na wielu innych platformach, na których dostępny jest kompilator języka C. Kod źródłowy interpretera i bibliotek języka jest otwarty (co oznacza, że m.in. można go pobrać i modyfikować bez ponoszenia opłat). Dostępność kodu źródłowego umożliwia również samodzielne jego dostosowywanie do działania na każdej nowej platformie, na której interpreter nie został jeszcze zaimplementowany (tzw. *portowanie*). Niezależność od implementacji systemowo-sprzętowej oznacza w tym przypadku to, że program napisany w języku Python w jednym systemie operacyjnym, działającym na określonej architekturze sprzętowej będzie można uruchomić w całkowicie odmiennym środowisku programowo-sprzętowym praktycznie bez zmian (lub z bardzo niewielkimi zmianami), uzyskując ten sam wynik działania algorytmu [2].

II. Algorytm zstępujący

W referacie przedstawiono wybrane elementy opracowanego pakietu w zakresie interpretacji języka Python prowadzącej do powstania drzewa zależności pomiędzy poszczególnymi składnikami postawionego problemu, w szczególności implementację algorytmu zstępującego TDA (Top-Down Algorithm) [2].



Rys.1. Algorytm *zstępujący* (przykład *drzewa problemu*)

Zadaniem algorytmu zstępującego jest przeglądanie problemu zadanego w postaci drzewa (Rys.1). Każdy problem może się składać z określonej ilości podproblemów, po skompletowaniu których możliwe jest oznaczenie go jako rozwiązanego. Ponieważ podproblemy mogą składać się z dużej ilości innych

[1] W tekście określenia serwer końcowy oraz węzeł końcowy reprezentują informatyczny element wykonawczy pakietu, odpowiednio w schematach funkcjonalnych oraz topologii przepływu informacji.

podproblemów w układzie relacji iteracyjnych – przeglądanie drzewa w czasie rzeczywistym podczas wykonywania zadania może się okazać nieefektywne dla bardziej złożonych zagadnień. W ogólności przeglądanie drzewa ma na celu odszukanie węzłów (podproblemów), które nie wykazują między sobą zależności i tym samym mogą być wykonywane równorzędnie na różnych jednostkach przetwarzających (procesorach). Tak wcześniej przygotowane listy wzajemnych uwarunkowań mogą również w przyszłości ułatwiać zaprogramowanie algorytmu adaptacyjnego, który w zależności od warunków wykonania będzie „decydował", które zadanie zrealizować jako kolejne, które ponowić i w jaki sposób złożyć finalne rozwiązanie problemu.

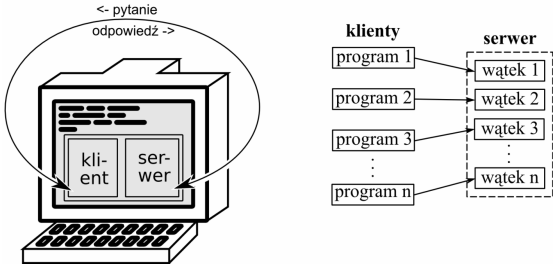
III. Projekt systemu informatycznego

W zakresie realizacji modelu przepływu informacji przyjęto w projekcie system przetwarzania rozproszonego bazującego na interakcyjnym współdziałaniu aplikacji według schematu funkcjonalności komponentów typu klient-serwer. Jako sieciową warstwę transportową wykorzystano dla połączeń WAN i LAN zestaw standardowych protokołów komunikacyjnych TCP/IP (Transmission Control Protocol / Internet Protocol).

Opracowany system analizująco-przetwarzający składa się z czterech podstawowych modułów – aplikacji sieciowej PyDist: serwer (węzeł) końcowy, magazyn danych, zarządzanie listą serwerów końcowych oraz asynchroniczne wykonywanie zadań, serwer HTTP umożliwiający monitorowanie stanu węzła końcowego. Poprawne działanie wszystkich komponentów zostało sprawdzone w systemach operacyjnych najczęściej wykorzystywanych w laboratoriach dydaktycznych tj. Linux i Windows instalowanych na architekturze sprzętowej x86 [2].

A. Komunikacja procesów

We wszystkich programach wykorzystywane są funkcje z pakietu RPyC, które dostarczają wielowątkowy serwer wykonujący polecenia, oraz warstwę komunikacji: program użytkownika – serwer końcowy sieci (Rys.2).

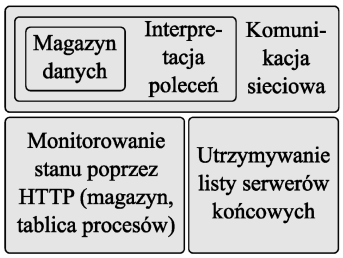


Rys.2. Realizacja połączeń sieciowych w obrębie jednego komputera

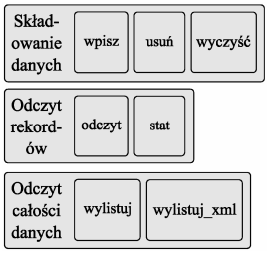
Serwer nasłuchuje na porcie TCP/18812 komputera na połączenia aplikacji klienckich oraz na porcie UDP/18813 na pakiety wysyłane na adres rozgłoszeniowy (broadcast) w sieci lokalnej - jeżeli zestawienie komponentów obejmuje również sieć LAN. Klient łączy się na port 18812 adresu IP określonego w parametrach konfiguracyjnych lub odszukanego poprzez rozgłoszenie, po czym następuje inicjalizacja komunikacji sieciowej. W przestrzeni nazw programu klienckiego pojawia się referencja do obiektu-pośrednika (*proxy*), za pomocą którego następuje interakcja ze zdalnym interpreterem języka Python. Obiekt-pośrednik jest to obiekt „udający" lokalnie zachowanie zdalnego obiektu. Posiada wszystkie metody i właściwości zdalnego obiektu, ale w momencie próby dostępu do którejś z nich następuje odwołanie się pośrednika poprzez kanał transportu sieciowego do prawdziwego obiektu i pobranie wartości z właściwości obiektu (object property) lub wyniku działania dedykowanej metody. Na podanej powyżej zasadzie działa dwustronna wymiana danych typu pytanie-odpowiedź pomiędzy wszystkimi komponentami pakietu w ramach dowolnych połączeń sieciowych. Struktura danych nie jest ograniczona specjalną specyfikacją i może odnosić się np. do przesyłania kolejnych instrukcji programu do wykonania, odbierania wyników obliczeniowych, czy też sprawdzania statusu danego węzła w sieci (czy jest wolny lub zajęty, itp.).

B. Węzeł (serwer) końcowy

Węzłem końcowym (zadaniowym) sieci nazwany jest element funkcjonalny (nazwa końcowy nie jest dyktowana przez topologię sieci, lecz charakter wykonywanych operacji) - program PyDist, w szczególności czekający na polecenia i wykonujący zadania. W skład jednego programu wchodzi: obsługa poleceń za pomocą modułu PyDist, serwer wykrywania *hostów* w sieci lokalnej poprzez adres rozgłoszeniowy sieci z użyciem protokołu UDP, instancja magazynu sieciowego (każdy serwer końcowy posiada swoją, na której można operować za pomocą zdalnych poleceń) oraz serwer HTTP pozwalający na monitorowanie stanu serwera końcowego (ilość procesów, status, itp.) oraz przeglądanie zawartości jego magazynu.



Rys.3. Moduły węzła (serwera) końcowego



Rys.4. Magazyn danych każdego węzła końcowego

Literatura

[1] Beazley D.M.:Python. Warszawa: RM 2002.
[2] Noszczyński M.: Modelowanie procedur przetwarzania rozproszonego w sieciach informatycznych laboratoriów dydaktycznych uwzględniających niezależność semantyki języka symbolicznego od implementacji systemowo-sprzętowej. Magisterska Praca Dyplomowa, Politechnika Wrocławska, IPEiE, Wrocław 2006.

Modelling the procedures of distributed processing in the network computers of didactic laboratories

Abstract - The adaptation of the system of the distributed processing was presented in the article. Worked out computer packets were talked over for student laboratories. The advantages of the programming in the language Python were underlined. The implementation of the Top-Down Algorithm was proposed to solving complex problems.